# XML データベースへの関数従属性を用いた推論攻撃に対する無限安全性検証法の提案

川居　裕人† 　橋本　健二†† 　石原　靖哲† 　藤原　融†

† 大阪大学大学院情報科学研究科　〒 565-0871 大阪府吹田市山田丘 1-5
†† 奈良先端科学技術大学院大学情報科学研究科　〒 630-0192　奈良県生駒市高山町 8916-5
E-mail: †{h-kawai,ishihara,fujiwara}@ist.osaka-u.ac.jp, ††k-hasimt@is.naist.jp

**あらまし**　推論攻撃とは，ユーザが許可されている問合せとその結果から, 機密情報の値の候補を絞り込む攻撃である．候補が有限個に絞りこめないとき，無限安全であるという．関数従属性をもつデータベースに対する推論攻撃の無限安全性に関して，先行研究では，ある前提条件下での検証法を提案している．本稿では，その前提条件をおかない場合にも対応した無限安全性検証法を提案する．

**キーワード**　データベースセキュリティ, XML, 推論攻撃, 関数従属性

# A Proposal of an Infinity Secrecy Verification Method Against Inference Attacks Using Functional Dependencies on XML Databases

Hiroto KAWAI†, Kenji HASHIMOTO††, Yasunori ISHIHARA†, and Toru FUJIWARA†

† Graduate School of Information Science and Technology, Osaka University,
1-5, Yamadaoka, Suita, Osaka, 565-0871, Japan
†† Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
E-mail: †{h-kawai,ishihara,fujiwara}@ist.osaka-u.ac.jp, ††k-hasimt@is.naist.jp

## 1. ま え が き

### 1.1 背　　景

近年, 様々な組織で少なからず機密性の高い情報が保持されているが, そのような機密情報が格納されているデータベースシステムにおいて, セキュリティ面での管理が必ずしも的確に行われていないことが問題となっている．データベースでの機密情報の漏洩を防ぐ手法として問合せにおけるアクセス制御がある．この手法では, データベース全体やユーザごとに許可する問合せと許可しない問合せを決め, 許可された問合せの答えのみを返す．これにより, 機密情報への直接のアクセスを制限し漏洩を防ぐ．しかし, アクセス制御により機密情報への直接のアクセスを禁止していたとしても, 推論攻撃を使用することで, 機密情報を絞り込まれたり, 特定されたりする可能性がある．データベースにおける推論攻撃とは, ユーザーが許可されている問合せとその実行結果から, 許可されていない問合せの実行結果 (機密情報) を推論し, 得ようとすることである．

［例 1.1］ XML データベースにおける推論攻撃の例を示す．$D$ は患者名と部屋, 病名を表す XML 文書であり, 以下のスキーマにしたがっているとする．

病院名 → 患者*
患者 → 部屋, 氏名, 病名
部屋 → string
氏名 → string
病名 → string

すなわち, 病院名要素の子には患者要素が 0 個以上並び, 各患者要素の子には部屋要素, 氏名要素, 病名要素が 1 つずつ並ぶ. 部屋要素, 氏名要素, 病名要素は文字列 (string) を値としてとる．$T_1$ は全患者の名前と部屋番号を, $T_2$ はある病名 (ここでは白血病) にかかっている患者の人数を, それぞれ取り出す問合せとする．「ある特定の患者の氏名と病名の組」を機密情報とし, $T_S$ は患者の氏名と病名を取り出す問合せとする．機密情報を守るため $T_S$ は実行を禁止し, $T_1$, $T_2$ のみ実行が許可されている．

それぞれの問合せの実行結果 $T_1(D)$, $T_2(D)$ がそれぞれ図 1, 2, に示すとおりであったとする. このとき, 問合せ $T_1$ の実行結果から, 全患者の氏名とその部屋番号が分かり, 問合せ $T_2$ の実行結果から, 白血病にかかっている患者の人数が分かる. しかし, これらの問合せからでは機密情報である, 患者 A, B, C のどの二人が白血病にかかっているかは特定することができない.

ここで $D$ が関数従属性「同室の患者は病名も同じである」を満たしているという情報を攻撃者が推論に用いるものとする. すると, 問合せ $T_1$ と $T_2$ の結果から, この関数従属性を満たす $D$ においては「患者 A と患者 C が白血病にかかっている」という可能性しかなく, 機密情報が特定されてしまう.

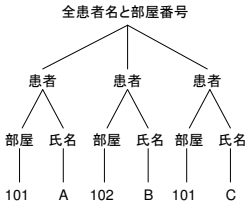



図 1　問合せ $T_1$ の実行結果

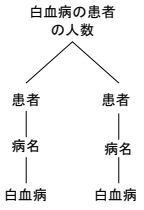



図 2　問合せ $T_2$ の実行結果

この例からも分かるように機密情報への直接の問合せを禁止していたとしても, 推論攻撃により機密情報が特定され漏洩する可能性がある. 推論攻撃によりどのような情報が得られるかは一般に自明でないため, データベース管理者としては, 安全性確保のために, 推論攻撃によって機密情報が漏洩する可能性をあらかじめ把握しておくことが重要であるといえる.

### 1.2　本研究の位置づけ

XML データベースへの推論攻撃に対する研究として文献 [1], [2], [3], [4], [5] がある. 文献 [1] では, XML データベースへの推論攻撃への対策として, XML 文書中にある機密情報を推論するための手がかりとなる要素に手を加えることで, 各ユーザへ推論に対して安全な security view を提供するという方法を提案している. security view とは, ユーザがアクセス権を持つ情報をちょうど含んだ文書と, その文書のスキーマ (ビュースキーマ) とからなるビューである. また文献 [2] では, 周知の知識を用いての機密情報への推論攻撃を対象としている. そして, 推論攻撃によって機密情報の漏洩の起きない最大部分文書を求めるアルゴリズムと, それに基づくシステム XGuard が提案されている. 提案されているアルゴリズムは次の 3 つのステップから構成されている.

- 推論を可能にする要素や値をオリジナルの XML 文書から発見する.
- それらのうち推論攻撃を不可能にするために最小の個数の要素や値を計算する.
- それらのみを削除することで, 最大部分文書を求める.

しかし, これらの提案方法においては, ユーザに対して許す問合せがただ 1 つということから, データベースの可用性が低くなるという欠点が挙げられる.

そこで文献 [3], [4], [5] では, XML データベースと許可されている複数の問合せから, そのデータベースが推論攻撃に対して安全であるかどうかを定式化し, その検証法を提案している. [4], [5] では, 関数従属性をもつ XML データベースへの推論攻撃に対する安全性検証法を提案しているが, [4] の検証法ではスキーマに再帰が存在しないという前提条件下で, また [5] の検証法では安全なものを見逃す可能性を許容できるという前提条件下でしか使用できないという欠点が存在する.

本稿では, 文献 [4] をベースにして, これらの前提条件をおかない場合にも対応した無限安全性検証法を提案する. 以降では今回提案する検証法で使用する木オートマトンや関数従属性の定義と検証法の大まかな流れを述べる.

## 2. 諸　定　義

本検証法で使用するモデルの定義は [4] と同じものを使用する. そのため本節では検証法で重要となる木オートマトン, 問合せモデル, 関数従属性の 3 つの定義を述べる.

### 2.1　XML スキーマのモデル

[定義 2.1]　非決定性有限木オートマトンは次の 4 つ組 $A = (Q, \Sigma, q_0, R)$ である.

- $Q$:状態の有限集合.
- $\Sigma$:有限アルファベット.
- $q_0 \in Q$:初期状態.
- $R$:遷移規則の集合. ここで $q \in Q, a \in \Sigma$ として, $e$ を $Q$ 上の言語を表現する正規表現とすると, 遷移規則は $(q, a, e)$ の形をとる. 特に, $R$ 中の任意の二つの遷移規則 $r_1 : (q_1, a_1, e_1)$, $r_2 : (q_2, a_2, e_2)$ で $L(e_1) \cap L(e_2) \neq \emptyset \wedge a_1 = a_2 \Rightarrow q_1 = q_2$ が成り立つとき, 決定性有限木オートマトンと呼ぶ. また, $A$ によって受理される全ての木の集合を $TL(A)$ と書く.

### 2.2　XML 文書への問合せモデル

XML 文書への問合せモデルについて述べる. XML 文書への問合せモデルとして, 決定性 relabeling 木変換器と決定性 deleting 木変換器という 2 種類の決定性木変換器の合成を採用している. 決定性 relabeling 木変換器と決定性 deleting 木変換器について定義を述べる.

#### 2.2.1　決定性トップダウン relabeling 木変換器

[定義 2.2]　決定性トップダウン relabeling 木変換器は, 次の 4 つ組 $T^T = (Q^T, \Sigma, q_0^T, R^T)$ である.

- $Q^T$: 状態の有限集合.
- $\Sigma$: アルファベット.
- $q_0^T \in Q^T$: 初期状態.
- $R^T$: 変換規則の集合. ただし, $q, q' \in Q^T$, $a, a' \in \Sigma$ とすると, 変換規則は $(q, a) \to a'(q')$ の形式である.

$T^T = (Q^T, \Sigma, q_0^T, R^T)$ の動作について述べる. $((q, a) \to a'(q')) \in R^T$ ならば, $T^T$ は $q(a(t_1 \cdots t_n))$ を $a'(q'(t_1) \cdots q'(t_n))$ へ変換可能であると定義する. 入力木 $t \in \mathcal{T}_\Sigma$ に対して, $q_0^T(t)$ から始めて, 以上のような変換をトップダウンに繰り返し, 最終的に $t' \in \mathcal{T}_\Sigma$ へ変換可能であるならば, $T^T$ は $t'$ を出力すると定義する ($T^T(t) = t'$ と書く).

#### 2.2.2　決定性ボトムアップ relabeling 木変換器

[定義 2.3]　決定性ボトムアップ relabeling 木変換器は以下の

4 つ組 $T^B = (Q^B, \Sigma, q_f^B, R^B)$ である.

- $Q^B$ : 状態の有限集合.
- $\Sigma$ : アルファベット.
- $q_f^B \in Q^B$ : 受理状態.
- $R^B$ : 変換規則の集合. ただし $q' \in Q^B$, $a, a' \in \Sigma$ とし, $e$ を $Q^B$ 上の正規表現とすると, 変換規則は $a(e) \to q'(a')$ の形式である.

$T^B = (Q^B, \Sigma, q_f^B, R^B)$ の動作について述べる. $t = a(q_1(t_1)\cdots q_n(t_n))$ とする. ここで, $q_1, \ldots, q_n \in Q$, $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$ である. $q_1 \cdots q_n \in L(e)$ でかつ $(a(e) \to q'(a')) \in R^B$ ならば, $T^B$ は $t$ を $q'(a'(t_1 \cdots t_n))$ へ変換可能であると定義する. 入力木 $t \in \mathcal{T}_\Sigma$ に対して, $t$ から始めて, 以上のような変換をボトムアップに繰り返し, 最終的に $q_f^B(t')$(ただし $t' \in \mathcal{T}_\Sigma$) へ変換可能であるならば, $T^B$ は $t'$ を出力すると定義する ($T^B(t) = t'$ と書く).

**2.2.3** 決定性 deleting 木変換器

[定義 2.4] 決定性 deleting 木変換器は以下の 4 つ組 $T^{del} = (Q^{del}, \Sigma, q, R^{del})$ である.

- $Q^{del} = \{q, q_\$\}$: 状態記号の集合.
- $\Sigma$: アルファベット.
- $R^{del}$: 変換規則の集合. ただし変換規則は, 以下の形式である.
  - $(q, a) \to a(q)$, $(a \in \Sigma, a \neq \sharp$ かつ, $a \neq \$$ のとき )
  - $(q, \sharp) \to q$
  - $(q, \$) \to q_\$$
  - $(q_\$, a) \to q_\$$, $(\forall a \in \Sigma)$

決定性 deleting 木変換器 $T^{del}$ は, 木を根頂点からトップダウンにたどり, $\sharp$ とラベル付けされているノードを削除する機能と, $\$$ とラベル付けされているノード以下の部分木を削除する機能を持つものである.

### 2.3 XML データベースにおける関数従属性

[定義 2.5] あるアルファベット $\Sigma$ について, $\Sigma$ 上のパス表現の集合 $Path_\Sigma$ を, 以下の構文で表現される集合と定義する.

$$Path_\Sigma ::= \text{`} \; / \; \text{'}a \mid \text{`} \; / \; \text{'}a \; Path_\Sigma \quad (a \in \Sigma)$$

パス表現は,‘ / ’(child 軸) と $\Sigma$ の要素から構成される. このパス表現のクラスは, XPath [6] の部分クラスである. また, 木におけるパス表現の値を, そのパス表現が指定する要素に対応する各頂点以下の部分木の集合と考える.

[定義 2.6] ある木 $t = a(t_1, \ldots, t_n) \in T_\Sigma$ において, パス表現 $p \in Path_\Sigma$ によって指定される値 $p(t)$ を以下のように定義する.

- $p = /x \quad (x \in \Sigma)$ のとき

$$/x\,(t) := \begin{cases} \{t\} & (x = a) \\ \emptyset & (x \neq a) \end{cases}$$

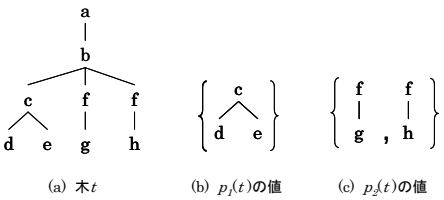
- $p = /x\,p' \quad (x \in \Sigma,\ p' \in Path_\Sigma)$ のとき

$$/x\,p'\,(t) := \begin{cases} \bigcup_{i=1}^{n} p'\,(t_i) & (x = a) \\ \emptyset & (x \neq a) \end{cases}$$

[定義 2.7] $p_1, \ldots, p_n \in Path_\Sigma$ とするとき, 木 $t \in T_\Sigma$ における, パス表現系列 $[p_1, \ldots, p_n]$ の値 $[p_1, \ldots, p_n](t)$ を以下のように定義する.

$$[p_1, \ldots, p_n](t) = \{s_1 \cdots s_n \mid s_1 \in p_1(t), \ldots, s_n \in p_n(t)\}$$

[例 2.1] 木 $t = a(b(c(de)f(g)f(h))$ において, パス表現 $p_1 = /a/b/c$, $p_2 = /a/b/f$ の値は, それぞれ $p_1(t) = \{c(de)\}$, $p_2(t) = \{f(g), f(h)\}$ である (図 3). このとき, $t_1 = c(de)$, $t_2 = f(g)$, $t_3 = f(h)$ とおくと, $[p_1, p_2](t) = \{t_1 t_2, t_1 t_3\}$ である.



(a) 木 $t$　(b) $p_1(t)$の値　(c) $p_2(t)$の値

図 3 パス表現が指定する値

[定義 2.8] XML における関数従属性の集合 $FD$ を, 以下のように表現される集合と定義する.

$$FD ::= (H, [x_1, \ldots, x_n], [y_1, \ldots, y_m])$$
$$(H, x_1, \ldots, x_n, y_1, \ldots, y_m \in Path_\Sigma)$$

[定義 2.9] 木 $t$ と関数従属性 $f = (H, [x_1, \ldots, x_n], [y_1, \ldots, y_m])$ $(f \in FD)$ について, 以下の式が成り立つとき, 木 $t$ が表す XML 文書は関数従属性 $f$ を満たすと定義する.

$$\forall u, v \in H(t), [x_1, \ldots, x_n](u) \cap [x_1, \ldots, x_n](v) \neq \emptyset$$
$$\Rightarrow [y_1, \ldots, y_m](u) \cap [y_1, \ldots, y_m](v) \neq \emptyset$$

関数従属性 $f$ が与えられたとき, $f$ を満たす木の集合を $TL(f)$ と書く.

## 3. 検証法の流れ

本節では, 提案する検証法の流れを説明する. 入力は許可されている各問合せ $T_i (i = 1, \ldots, n)$ とその実行結果 $T_i(D)$, 与えられているデータベーススキーマ $A_G$, データベースが持つ関数従属性 $f (\in FD)$, 機密情報への問合せ $T_S$ とし, 出力は無限安全かどうか, すなわち $\{T_S(D') \mid D' \in TL(f) \cap TL(A_G), T_1(D) = T_1(D'), \ldots, T_n(D) = T_n(D')\}$ の要素が無限かどうかとする.

本検証法は以下の二つのステップからなる.

(1) $T_i$, $T_i(D)$, $A_G$ から元の XML 文書候補集合を計算する.

(2) XML 文書候補集合と $T_S$, $f$ から無限安全性を判定する.

以降, 各ステップ内で行う操作について述べる.

### 3.1 ステップ (1): XML 文書候補集合の計算

このステップでは, 入力から, 元の XML 文書 $D$ の候補を受理する木オートマトン $A_D$ を構成する. すなわち, $TL(A_D) = \{D' \in TL(A_G) \mid T_i(D') = T_i(D)\ (1 \leqq i \leqq n)\}$ を満たす $A_D$ を構成する. $A_D$ の構成法は文献 [4] の方法を使用する.

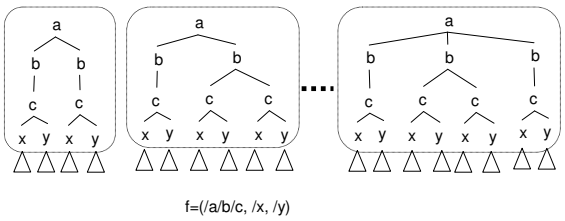



図 4　$TL(A_S^{AD})$ の要素の例

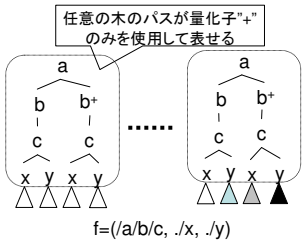



図 5　$TL(A_x^{AD})$ の要素の例

### 3.2　ステップ (2): 無限安全性の判定

$A_D$ と $T_S$ から $TL(A'_S) = \{T_S(t) \mid t \in TL(A_D)\}$ を満たす木オートマトン $A'_S$ を構成するとする．このとき, 仮に $A_D$ が $f$ を満たす木のみを受理する木オートマトンである, つまり $TL(A_D) = TL(A_D) \cap TL(f)$ が成立するならば, $TL(A'_S)$ の要素数から無限安全性の判定を行える．しかし, 一般には $TL(A_D) = TL(A_D) \cap TL(f)$ は成立しない．なぜなら, ステップ (1) では関数従属性を考慮せずに $A_D$ を構成しているからである．関数従属性を考慮しないのは, 一般には関数従属性を満たす木のみを受理する有限木オートマトンは構成できないことが知られているためである．そこで, $TL(A_D) \cap TL(f)$ を陽に求めることなく, 無限安全性の判定を行う方法を与える．

#### 3.2.1　検証法のアイデア

機密情報問合せ $T_S$ は relabeling 木変換器 $T_{Sr}$ と deleting 木変換器 $T_{Sd}$ の合成で定義されている．$A_D$ と $T_S$ から $TL(A_S) = \{(t,t') \mid t' = T_{Sr}(t), t \in TL(A_D)\}$ を満たす木オートマトン $A_S$ を構成する．また, $TL(A_S)$ における第一成分の集合を $TL(A_S^{AD}) = \{t \mid (t,t') \in TL(A_S)\}$ と定義する．このとき, $(t,t') \in TL(A_S)$ について, $t \in TL(f)$ であるならば $T_{Sd}(t')$ は機密情報候補となる．つまり $K = |\{T_{Sd}(t') \mid (t,t') \in TL(A_S), t \in TL(f)\}|$ とすると, $K = \infty$ かどうかが判定可能であれば無限安全性の判定も可能である．しかし, $A_S$ では遷移規則中の正規表現の disjunction 演算の出現に関して何の制約もないため, $A_S^{AD}$ は図 4 のように関数従属性のパスの形が様々な木を受理する．$K = \infty$ を判定するためには, そのパスの形による場合分けが必須であるため, 判定手順を正確に与えることは困難である．しかし, 図 5 のように関数従属性のパスの形が固定された木オートマトン $A_x$ では, 後で述べるように $K_x = |\{T_{Sd}(t') \mid (t,t') \in TL(A_x), t \in TL(f)\}| = \infty$ かどうかの判定手順を比較的容易に与えることができる．そこで, 検証法は「$K = \infty \Leftrightarrow$ 関数従属性のパスの形が固定された $A_{S1}, \ldots, A_{Sn}$ の少なくとも 1 つの $A_{Si}$ について $K_{Si} = \infty$」を満たすような $A_{S1}, \ldots, A_{Sn}$ を $A_S$ から構成するという方針をとる．具体的には以下の手順をとる．

i　$A_S$ から, 関数従属性のパスを導出する遷移規則中の正規表現から disjunction 演算を除去し, 関数従属性のパスの形が固定された木オートマトン $A_{S1}, \ldots, A_{Sn}$ を構成する．

ii　各 $A_{Si}(1 \leqq i \leqq n)$ で $K_{Si} = \infty$ かどうかを判定する．

#### 3.2.2　disjunction 演算の除去

disjunction 演算を除去するために, 木オートマトンがもつ遷移規則集合を, 常に関数従属性のパスを生成する遷移規則集合 $R_f$ と, 常にそれ以外の部分を生成する遷移規則集合 $R_{nf}$ に分けたい．しかし一般には $A_S$ の遷移規則集合をそのように分けることはできない．そこで $A_S$ に次のような操作を順に行い, $TL(A_S) = TL(A_{S_B})$ である木オートマトン $A_{S_B}$ を構成する．

1　与えられた関数従属性 $f$ から, 任意の木を受理する木オートマトン $A_f$ を構成する．ただし, 関数従属性のパスが持つ状態はそれ以外に使用しないように状態を割り振り構成する．

2　構成した各木オートマトン $A_f$ と $A_S$ との交わりをとり, $A_{S \cap f}$ を構成する．

3　$A_{S \cap f}$ を等価なボトムアップ決定性有限木オートマトン $A_{S_B}$ に変換する．

**関数従属性 $f$ から木オートマトンを構成する方法**

ここでは, 与えられた 1 つの関数従属性 $f = (/h_1/\ldots/h_l\ (= H), /x_1/\ldots/x_m\ (= X), /y_1/\ldots/y_n\ (= Y))$ から任意の木を受理しかつ, 関数従属性のパスに割り振られる状態がそれ以外では割り振られない木オートマトン $A_f = (Q, \Sigma, Q_s, R_f)$ の構成を述べる．ここで関数従属性 $f = (/H, /X, /Y)$ において $X = /Y/\cdots/x_m$ と表せるとき, パス $X$ はパス $Y$ を包含しているといい, $X \supseteq Y$ と書く．構成法は X と Y に包含関係が無い場合, $X \supseteq Y$ の場合, $X \subset Y$ の場合の三つの場合で異なるが, ここでは紙面の都合上 X と Y に包含関係が無い場合での構成法のみを書く．

$Q$ は以下の 4 種類の状態の集合である．

- 関数従属性のパスに割り振られる状態.

$\{q_{path} \mid path \in \{h_1, \ldots, h_l, x_1, \ldots, x_m, y_1, \ldots, y_n\}\}$

- 関数従属性のパスと同じラベルを導出するが, パスにならない状態.

$\{\bar{q}_{path} \mid path \in \{h_1, \ldots, h_l, x_1, \ldots, x_m, y_1, \ldots, y_n\}\}$

- 関数従属性のパスのラベルを導出しない状態.

$\{q_{\overline{path}} \mid path \in \{h_1, \ldots, h_l, x_1, \ldots, x_m, y_1, \ldots, y_n\}\}$

- 任意の木を生成する状態.

$\{q_{any}\}$

初期状態 $Q_s$ は $\{q_{h1}, \bar{q}_{h1}, q_{\bar{h}1}\}$ である．

**X と Y に包含関係が無い場合**

- $1 \leqq i \leqq l-1$

- $q_{hi} \to h_i((q_{h(i+1)}|\bar{q}_{h(i+1)}|q_{\bar{h}(i+1)})^* q_{h(i+1)}$ $(q_{h(i+1)}|\bar{q}_{h(i+1)}|q_{\bar{h}(i+1)})^*)$
- $\bar{q}_{hi} \to h_i(\bar{q}_{h(i+1)}|q_{\bar{h}(i+1)})^*$
- $q_{\bar{h}i} \to \alpha(q_{any})^*$ $(\alpha \in \Sigma - \{h_i\})$
- $q_{hl} \to h_l((q_{x1}|\bar{q}_{x1}|q_{y1}|\bar{q}_{y1}|q_{\overline{x1,y1}})^* q_{x1}$ $(q_{x1}|\bar{q}_{x1}|q_{y1}|\bar{q}_{y1}|q_{\overline{x1,y1}})^*)$
- $\bar{q}_{hl} \to h_l(\bar{q}_{x1}|q_{\bar{x}1})^*$
- $q_{\bar{h}l} \to \alpha(q_{any})^*$ $(\alpha \in \Sigma - \{h_l\})$
- $q_{\overline{x1,y1}} \to \alpha(q_{any})^* (\alpha \in \Sigma - \{x_1, y_1\})$
- $1 \leqq i \leqq m-2$
- $q_{xi} \to x_i((q_{x(i+1)}|\bar{q}_{x(i+1)}|q_{\bar{x}(i+1)})^* q_{x(i+1)}$ $(q_{x(i+1)}|\bar{q}_{x(i+1)}|q_{\bar{x}(i+1)})^*)$
- $\bar{q}_{xi} \to x_i(\bar{q}_{x(i+1)}|q_{\bar{x}(i+1)})^*$
- $q_{\bar{x}i} \to \alpha(q_{any})^*$ $(\alpha \in \Sigma - \{x_i\})$
- $q_{x(m-1)} \to x_{m-1}((q_{xm}|q_{\bar{x}m})^* q_{xm}(q_{xm}|q_{\bar{x}m})^*)$
- $\bar{q}_{x(m-1)} \to x_{m-1}(q_{\bar{x}m})^*$
- $q_{\bar{x}(m-1)} \to \alpha(q_{any})^*$ $(\alpha \in \Sigma - \{x_{m-1}\})$
- $q_{xm} \to x_m(q_{any})^*$
- $q_{\bar{x}m} \to \alpha(q_{any})^*$ $(\alpha \in \Sigma - \{x_m\})$
- $q_{y1} \to y_i((q_{y2}|\bar{q}_{y2}|q_{\bar{y}2})^* q_{y2}(q_{y2}|\bar{q}_{y2}|q_{\bar{y}2})^*)$
- $\bar{q}_{y1} \to y_1(\bar{q}_{y2}|q_{\bar{y}2})^*$
- $2 \leqq i \leqq n-2$
- $q_{yi} \to y_i((q_{y(i+1)}|\bar{q}_{y(i+1)}|q_{\bar{y}(i+1)})^* q_{y(i+1)}$ $(q_{y(i+1)}|\bar{q}_{y(i+1)}|q_{\bar{y}(i+1)})^*)$
- $\bar{q}_{yi} \to y_i(\bar{q}_{y(i+1)}|q_{\bar{y}(i+1)})^*$
- $q_{\bar{y}i} \to \alpha(q_{any})^*$ $(\alpha \in \Sigma - \{y_i\})$
- $q_{y(n-1)} \to y_{n-1}((q_{yn}|q_{\bar{y}n})^* q_{yn}(q_{yn}|q_{\bar{y}n})^*)$
- $\bar{q}_{y(n-1)} \to y_{n-1}(q_{\bar{y}n})^*$
- $q_{\bar{y}(n-1)} \to \alpha(q_{any})^*$ $(\alpha \in \Sigma - \{y_{n-1}\})$
- $q_{yn} \to y_n(q_{any})^*$
- $q_{\bar{y}n} \to \alpha(q_{any})^*$ $(\alpha \in \Sigma - \{y_n\})$
- $q_{any} \to \alpha(q_{any})^*$ $(\alpha \in \Sigma)$

上記のようにして構成された $A_f$ は任意の木を受理するので, $A_f$ と $A_S$ との交わりをとっても $TL(A_S) = TL(A_{S\cap f})$ であることは自明である.

またこのような $A_{S\cap f}$ では, 状態を見るだけでその遷移規則が関数従属性のパスのどの部分で使用されるかが判断できる. また, $A_{S\cap f}$ から等価に変換されてできるボトムアップ決定性有限木オートマトン $A_{S_B}$ においても, 同様のことがいえる. そのため $A_{S_B}$ がもつ遷移規則集合を $R_f$ と $R_{nf}$ に分割できる. 以降では $A_{S_B}$ の遷移規則集合 $R_f$ で使われている disjunction 演算の除去の方法を述べる.

[補題 3.1] $A$ を遷移規則集合を $R_f$ と $R_{nf}$ に分割可能な木オートマトンとし, 関数従属性 $f$ が与えられているとする. このとき $r = (q, a, e), (r \in R_f)$ の $e$ 中にある部分表現 $(e_1|e_2)^+$ を $(e_1^+|e_2^+|e_1^+e_2^+)$ に置き換えた遷移規則を $r'$ とする. そして, 遷移規則 $r$ の代わりに $r'$ をもつ木オートマトンを $A'$ とする. 二つの木オートマトン $A$, $A'$ の間に次のことが成立する.

$$|TL(A) \cap TL(f)| = \infty \Leftrightarrow |TL(A') \cap TL(f)| = \infty$$

$e^*$ の表現は $(\varepsilon|e^+)$ と等価であるため, $e^*$ は disjunction 演算が含まれていると考えられる. そこで, まず初めに $r = (q, a, e), (r \in R_f)$ において, $e$ 中にある部分表現 $e^{'*}$ を $(\varepsilon|e^{'+})$ に置き換える. 次に $r = (q, a, e), (r \in R_f)$ の $e$ 中にある部分表現 $(e_1|e_2)^+$ を $(e_1^+|e_2^+|e_1^+e_2^+)$ に置き換える. $L((e_1|e_2)^+) \supset L(e_1^+|e_2^+|e_1^+e_2^+)$ であるが, 無限安全性は補題 3.1 から保存されていることがいえる. その後, $f = (H, [x_1, \ldots, x_n], [y_1, \ldots, y_m])$ の $x_1, \ldots, x_n, y_1, \ldots, y_m$ を導出する遷移規則中にある disjunction 演算を等価変換によって除去する. これによって $R_f$ の中ではパス $H$ を導出する遷移規則中にしか disjunction 演算が存在しなくなる. 最後にパス $H$ を導出する遷移規則中にある disjunction 演算の除去を次のことの繰り返しによって行う. 入力を $e$ が $(e_1 \mid e_2)$ の形の部分表現を含む遷移規則 $r = (q, a, e) \in R_f$ をもつ木オートマトンとすると, 出力は $A'_{s1} = (Q, \Sigma, q_s, R'_{f1} \cup R_{nf})$, $A'_{s2} = (Q, \Sigma, q_s, R'_{f2} \cup R_{nf})$ の二つの木オートマトンである. ただし, $R'_{f1}$, $R'_{f2}$ はそれぞれ $R'_{f1} = R_f - \{r\} \cup \{r_1\}$, $R'_{f2} = R_f - \{r\} \cup \{r_2\}$ である. また, $r_1$ は $e$ 中の一箇所の部分表現 $(e_1 \mid e_2)$ を $(e_1 \mid \emptyset)$ に置き換えた遷移規則, $r_2$ は $(\emptyset \mid e_2)$ に置き換えた遷移規則である.

この disjunction 演算の除去に関して次の補題が成立する.

[補題 3.2]

$$|TL(A_{S_B}^{AD}) \cap TL(f)| = \infty \Leftrightarrow$$
$$|TL(A_{s1}^{'AD}) \cap TL(f)| = \infty \text{ または } |TL(A_{s2}^{'AD}) \cap TL(f)| = \infty$$

$TL(A_{s1}^{'AD}) \cup TL(A_{s2}^{'AD})$ は $TL(A_{S_B}^{AD})$ の部分集合となるように構成されるため, $A_{s1}^{'AD}$, $A_{s2}^{'AD}$ のどちらかで $|TL(A_{si}^{'AD}) \cap TL(f)| = \infty$ が成り立つならば $|TL(A_{S_B}^{AD}) \cap TL(f)| = \infty$ は成立する. 逆は背理法で示せる. $|TL(A_{S_B}^{AD}) \cap TL(f)| = \infty$ かつ, $TL(A_{s1}^{'AD})$, $TL(A_{s2}^{'AD})$ はそれぞれ, 高々有限個の要素集合であると仮定する. そして, + 演算や再帰が $R_f$ にある場合と $R_{nf}$ にある場合に分け, それぞれで仮定に対する矛盾を導くことで補題を証明できる.

### 3.2.3 $|TL(A_{Si}^{AD}) \cap TL(f)| = \infty$ の判定法

$A_{S_B}$ から構成された木オートマトン $A_{Si}(1 \leqq i \leqq n)$ における $TL(A_{Si}^{AD})$ の要素は図 5 のように, 関数従属性のパスの形が固定されている. さらに, $R_{nf}$ に対しては変換操作を行っていないため, $R_{nf}$ においてはボトムアップ決定性が保持されていることがいえる.

関数従属性のパスの導出がただ一通りでかつ, パス以外の導出がボトムアップ決定性である木オートマトン $A_{Si}$ は, 次のような性質をもつことが分かっている.

[補題 3.3]

$$|TL(A_{Si}^{AD})| = \infty \Rightarrow |TL(A_{Si}^{AD}) \cap TL(f)| = \infty$$
$$\vee\ TL(A_{Si}^{AD}) \cap TL(f) = \emptyset$$

そのため, $TL(A_{Si}^{AD})$ に少なくとも 1 つ関数従属性を満たす木が存在することを示せると, 補題 3.3 から $|TL(A_{Si}^{AD}) \cap TL(f)| = \infty$ であることがいえる. また, このような木オートマトン $A_{Si}$ は次のような性質も成立する.

［補題 3.4］ $A_{Si}$ の遷移規則中にある $e^{*(+)}$ の繰り返し回数や再帰回数を高々$A_{Si}$ が持つ状態数 $|Q^{Si}|$ に抑えた木オートマトンを $A_{Si_{|Q|}}$ とおく．このとき，

$$TL(A^{AD}_{Si}) \cap TL(f) \neq \emptyset \Leftrightarrow TL(A^{AD}_{Si_{|Q|}}) \cap TL(f) \neq \emptyset$$

が成り立つ．

補題 3.4 から $TL(A^{AD}_{Si_{|Q|}})$ に関数従属性を満たす木が存在することを示すことで，$TL(A^{AD}_{Si})$ に少なくとも 1 つ関数従属性を満たす木が存在することを示せる．

$A_{Si_{|Q|}}$ は遷移規則中にある $e^{*(+)}$ の繰り返し回数や再帰回数を高々$A_{Si}$ が持つ状態数 $|Q^{Si}|$ に抑えた木オートマトンである．つまり，$TL(A_{Si_{|Q|}})$ は有限である．加えて，$TL(A^{AD}_{Si_{|Q|}}) = \{t \mid (t,t') \in TL(A_{Si_{|Q|}})\}$ であるため $TL(A^{AD}_{Si_{|Q|}})$ も有限である．つまり $TL(A^{AD}_{Si_{|Q|}})$ の各要素で関数従属性を満たしているかを調べることで，$TL(A^{AD}_{Si_{|Q|}}) \cap TL(f) \neq \emptyset$ かどうかは判定可能である．

最後に $K_{Si} = \infty$ であるかどうかを判定する方法を述べる．$K_{Si} = |\{T_{Sd}(t') \mid (t,t') \in TL(A_{Si}), t \in TL(f)\}|$ である．そのため $|TL(A^{AD}_{Si}) \cap TL(f)| \neq \infty$ ならばただちに $K_{Si} \neq \infty$ と判定できる．$|TL(A^{AD}_{Si}) \cap TL(f)| = \infty$ であるときこのような $(t,t')$ は無限個存在する．しかし，$(t,t')$ が無限個存在しても $|T_{Sd}(t')| = \infty$ であるとは限らない．なぜなら，異なる二つの $(t_1,t'_1)$，$(t_2,t'_2) \in TL(A_{Si}), t_1, t_2 \in TL(f)$ であっても，$T_{Sd}(t'_1) = T_{Sd}(t'_2)$ となる可能性があるためである．ここで $A_{Si}$ と $T_{Sd}$ から $TL(A_{del_{Si}}) = \{T_{Sd}(t') \mid (t,t') \in TL(A_{Si})\}$ を満たす木オートマトン $A_{del_{Si}}$ を構成でき，かつこのとき，$TL(A_{del_{Si}})$ と $K_{Si}$ には次のような関係があることが示せる．

［補題 3.5］

$|TL(A^{AD}_{Si}) \cap TL(f)| = \infty$ かつ $|TL(A_{del_{Si}})| = \infty \Leftrightarrow K_{Si} = \infty$

証明概要

$K_{Si} = \infty \Rightarrow |TL(A_{del_{Si}})| = \infty$ は定義から明らかである．逆については，$|TL(A^{AD}_{Si}) \cap TL(f)| = \infty$ かつ $K_{Si} \neq \infty$ としたときに，$T_S$ と $f$ の性質から $|TL(A_{del_{Si}})| \neq \infty$ が示せる．証明の詳細については紙面の都合上省略する．

補題 3.5 から，$|TL(A_{del_{Si}})| = \infty$ を示せば $K_{Si} = \infty$ が示せることが分かる．そこで，$|TL(A_{del_{Si}})| = \infty$ であるかどうかを判定する方法を述べる．

$A_{del_{Si}}$ は $TL(A_{del_{Si}}) = \{T_{Sd}(t') \mid (t,t') \in TL(A_{Si})\}$ を満たす木オートマトンである．そのため図 6, 7 から分かるように $A_{Si}$ の根頂点から葉までの経路のいずれかで“ $” が現れるより前に $e^{*(+)}$ の表現が存在すれば，$|TL(A_{del_{Si}})| = \infty$ が成立する．この判定は次のよう行う．まず，$A_{Si}$ の遷移規則を使用して $A_{Si}$ が受理する木の根から葉までの可能な経路をすべて導出する．この際，各経路の導出では同じ遷移規則を二回以上使用しないものとする．そして，このようにして導出された経路のいずれかで，ラベル“ $” をもつ状態に到達する前に，$e^{*(+)}$ の形の式に含まれる状態に到達可能であるとき，$|TL(A_{del_{Si}})| = \infty$ と判定する．再帰の場合についても同様にして判定する．

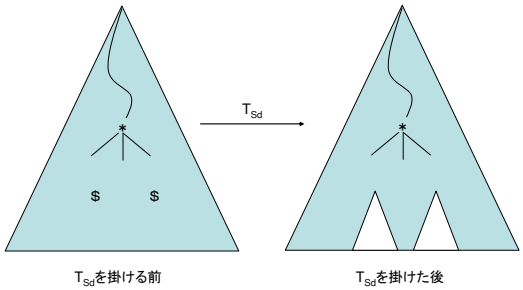



図 6 $|TL(A_{del_{Si}})| = \infty$ となる場合

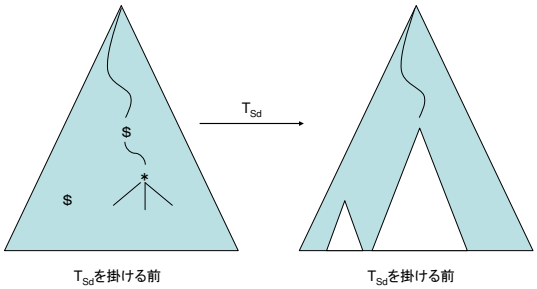



図 7 $|TL(A_{del_{Si}})| = \infty$ とならない場合

そのため，補題 3.5 を用いることで $K_{Si} = \infty$ が判定できる．したがって，以上から次の定理が得られる．

［定理 3.1］ ある $i$ について $TL(A^{AD}_{Si}) \cap TL(f) \neq \emptyset \wedge |TL(A_{del_{Si}})| = \infty$ のとき，かつそのときのみ，無限安全である．さらに，無限安全性は判定可能である．

## 4. あとがき

本稿では，関数従属性を用いた推論攻撃に対する無限安全性の検証法を提案した．今回定義している関数従属性は子供軸のみを許可している．さらに子孫軸などの軸を許可したとすると，関数従属性から目的の木オートマトンを作ることが困難になり，さらには，disjunction 演算の除去も困難になる．そのため関数従属性の表現能力を上げると単純に提案検証法を拡張するだけでは対応できないと予想している．関数従属性の表現能力を上げたときの検証法は提案検証法とは異なるアプローチで行う必要性が高いと考えられる．また，本検証法で無限安全ではないと判定されたとき，つまり機密情報候補が有限であるとき，その機密情報候補が具体的に何個であるかは分からない．そこで，今後は無限安全ではないときに機密情報候補を判定する方法を考えていく．

## 文　　献


[1] W.Fan, C.Y.Chan, and M.Garofalakis: “Secure XML Querying with Security Views,” Proc. 23rd SIGMOD, pp. 587-598, 2004.

[2] X.Yang and C.Li: “Secure XML Publishing without Information Leakage in the Presence of Data Inference,” Proc. 30th VLDB, pp. 96-107, 2004.

[3] K.Hashimoto, K.Sakano, F.Takasuka, Y.Ishihara and T.Fujiwara: “Verification of the security against inference attacks on XML databases, ” IEICE Transactions on Information and Systems, E92-D(5), pp. 1022-1032, 2009.

[4] 阪野 公秀, 橋本 健二, 石原 靖哲, 藤原 融, “XML データベースへの関数従属性を用いた推論攻撃に対する 安全性の定式化とある条件下での安全性検証法の提案,” コンピュータセキュリティシンポジウム 2008 論文集, pp. 461-466, 2008.

[5] 阪野 公秀, 橋本 健二, 石原 靖哲, 藤原 融, “XML データベースへの関数従属性を用いた推論攻撃に対する安全性の定式化と その検証法の提案,” データ工学と情報マネジメントに関するフォーラム, D4-2, 2009.

[6] “XML Path Language (XPath) Version 1.0,” http://www.w3.org/TR/xpath.